Patent Application

for

SYSTEM AND METHOD FOR GENERATING DATA VALIDATION RULES

by

Ernest S. Baugher

SYSTEM AND METHOD FOR GENERATING DATA VALIDATION RULES

BACKGROUND OF THE INVENTION

When businesses receive new data for their computer systems, they typically must validate that data before it can be processed. For example, when a bank receives a new loan application, the information contained in the application is examined and determined to be legitimate and complete before the application is processed. If the application fails validation, processing may halt and the data in the application may be corrected before processing can proceed.

There are many different types of validation. Validation logic may apply to a single data item. The validation logic can determine if a data value is the correct type (e.g. numeric, alphanumeric, text, etc.) and the correct length. The validation logic can also determine if the data value is a member of a specified set of values (e.g. a valid two character state code) or if it falls within a specified range of values. Alternatively, validation logic may involve multiple data items and the validity of the relationships between them. For example, validation logic might dictate that a loan amount cannot be greater than ten times the amount of equity. Validation logic can become arbitrarily complex. Some validations may be conditional (i.e. the application of the validation is dependent upon the successful execution of a conditional statement). For example, some loan validation rules might only apply to certain types of loans. The concept of validation encompasses any logic that examines one or more data items and determines a Boolean outcome of true or false.

The implementation of validation logic has a number of problems associated with it. First, the creation of validation logic is a time consuming and costly endeavor. Many systems contain hundreds or even thousands of validation rules. In current application development, the validation logic typically may not be created or tested independently from the application; i.e. the application must be executed in order to execute the validation logic. This provides a very inefficient means to create and test validation logic. In addition to the cost of creating validation logic, it is also difficult and costly to maintain validation logic. Since it is usually not well segregated from the other application logic, it cannot be managed and maintained as a separate IT asset. Over time

it typically becomes harder and more costly to maintain due to inadequate documentation, staff turnover, etc.. The implementation and maintenance of validation logic also suffers from a lack of business visibility into the process. Validation logic typically may be defined or approved by business users, yet there are few or no tools that make it possible for business users to participate directly in rule creation, testing, or maintenance. Their participation is usually limited to reviewing documentation that is not generated directly from the implementation of the validation logic. Such documentation is costly to maintain and over time often diverges from the implementation due to lack of diligence or resources.

Once created, validation logic may be difficult to reuse inside other applications within the enterprise and between enterprises. In the past, many large enterprises have relied on mainframes to be the sole domicile of their business logic. However, the internet has required the use of new application platforms such as J2EE and .NET, making it difficult for an enterprise to maintain their business logic on a single platform. Consequently, validation logic is often implemented and duplicated on multiple platforms within the enterprise. It would be desirable to create and manage validation logic for multiple platforms from a single point of control. In addition to the current difficulties associated with reusing validation logic within an enterprise, it may be difficult to share validation logic with external enterprises. The rise of E-Commerce has increased the amount of B2B interactions and the amount of data that is transmitted electronically between enterprises. These interactions are giving rise to new industry standards for exchanging information. These standards often include validation rules as well as data layout specifications. Yet each enterprise is responsible for its own implementation of the validation logic. This creates inefficiency, redundancy of effort, and costs associated with reconciling different implementations between enterprises. Therefore, it may be difficult for an enterprise to share its validation logic with other enterprises.

Finally, the current implementations of validation logic may not mitigate the cost associated with the manual investigation and correction of invalid data. When data fails validation it often may require manual intervention to investigate and correct the data (i.e. an individual has to research the cause of the failure and either correct the data or the

validation rules). This cost may be either borne by the company receiving the data, the company sending the data, or both.

SUMMARY OF THE INVENTION

The invention of the present disclosure features a method for generating validation logic executable in a target application in a target platform including creating a metarule, and creating target domain metarule validation logic based on the metarule and a target language of the target platform.

In another aspect, the invention of the present disclosure features a method including creating a metarule, creating a target domain attribute, creating a mapping of the metarule to the target domain attribute, and creating target domain attribute validation logic executable in a target application in a target platform based on the metarule, the mapping of the metarule to the target domain attribute, and the target language of the target platform.

In another aspect, the invention of the present disclosure features a method including creating a metarule, creating a meta domain attribute, associating the metarule with the meta domain attribute, creating a target domain attribute, mapping the meta domain attribute to the target domain attribute, and creating target domain attribute validation logic executable in a target application in a target platform based on the metarule, the meta domain attribute, the target domain attribute, the mapping between the meta domain attribute and the target domain attribute, the metarule associated with the meta domain attribute, and the target language of the target platform.

In another aspect, the invention of the present disclosure features a system including a meta domain editor for creating a meta domain attribute and an associated metarule, a target domain editor for creating a target domain attribute and for mapping the meta domain attribute to the target domain attribute, and a target domain attribute validation function generator for generating a target domain attribute validation function based on the meta domain attribute, the associated metarule, the target domain attribute, the mapping of the meta domain attribute to the target domain attribute, and the target language of the target platform.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG 1 is a diagram showing an overview of a system according to one aspect of the system of the present disclosure.

FIG 2 is a diagram illustrating the basic architecture of a system according to one aspect of the system of the present disclosure.

FIG 3 is a diagram illustrating the information contained in meta domains according to one aspect of the system of the present disclosure.

FIG 4 is a diagram illustrating the information contained in metarules and the association of metarules with meta domain attributes according to one aspect of the system of the present disclosure.

FIG 5 is a diagram illustrating the information contained in meta domain test suites according to one aspect of the system of the present disclosure.

FIG 6 is a diagram illustrating the information contained in meta domain documentation according to one aspect of the system of the present disclosure.

FIG 7 is a diagram illustrating the information contained in a Java target domain according to one aspect of the system of the present disclosure.

FIG 8 is a diagram illustrating the information contained in a mapping of a meta domain to a Java target domain according to one aspect of the system of the present disclosure.

FIG 9 is a diagram illustrating the information contained in a Java target domain test suite according to one aspect of the system of the present disclosure.

FIG 10 is a diagram illustrating the information contained in Java target domain documentation according to one aspect of the system of the present disclosure.

FIG 11 is a diagram illustrating a generic process flow for the creation of validation rules according to one aspect of the system of the present disclosure.

FIG 12 is a diagram illustrating an example meta domain according to one aspect of the system of the present disclosure.

FIG 13 is a diagram illustrating metarules for the example meta domain according to one aspect of the system of the present disclosure.

FIG 14 is a diagram illustrating an example Java target domain according to one aspect of the system of the present disclosure.

FIG 15 is a diagram illustrating a mapping between the example meta domain and the example Java domain according to one aspect of the system of the present disclosure.

FIG 16 is a diagram illustrating the process for generating target validation rules according to one aspect of the system of the present disclosure.

FIG 17 is a diagram illustrating the structure and execution sequence of the validation rules generated for the example Java domain according to one aspect of the system of the present disclosure.

FIG 18 is a diagram illustrating a test suite for the example meta domain according to one aspect of the system of the present disclosure.

FIG 19 is a diagram illustrating a test suite for the example Java domain according to one aspect of the system of the present disclosure.

FIG 20 is a diagram illustrating the process for executing test suites according to one aspect of the system of the present disclosure.

FIG 21 is a diagram illustrating the process for generating documentation according to one aspect of the system of the present disclosure.

FIG 22 is a diagram illustrating the documentation generated for the example meta domain according to one aspect of the system of the present disclosure.

FIG 23 is a diagram illustrating the documentation generated for the example Java domain according to one aspect of the system of the present disclosure.

FIG 24 is a diagram illustrating the process for using the system of the present disclosure within an enterprise according to one aspect of the system of the present disclosure.

FIG 25 is a diagram illustrating the process for using the system of the present disclosure between enterprises according to one aspect of the system of the present disclosure.

DETAILED DESCRIPTION OF THE INVENTION

The system of the present disclosure is based on the concept of metarules. A metarule may be an extension of the concept of metadata (i.e. data about data) and may be defined in a high level representation rather than in a specific programming language. Since metarules may not defined in a programming language they may not actually be

executed in production. Metarules may be used to derive functionally equivalent rules implemented in a specific programming language and executed in a production platform.

FIG 1 shows the Validator System 110, according to one aspect of the present disclosure, and multiple Target Platforms 120. The Validator System 110 is capable of generating Target Specific Validation Rules 111 for target applications associated with the Target Platforms 120. The Target Specific Validation Rules 111 are included in the target applications at development time. The target applications may execute on handheld computers, laptops, personal computers, workstations, or mainframes.

Each Target Platform 120 encompasses numerous potential target applications. Target Platforms may include, for example, the following programming languages: Cobol Platform 122, C Platform 123, C# Platform 124, C++ Platform 125, Java Platform 126, and Visual Basic Platform 129. In such cases, a target application may be any application written in the programming language of the platform that utilizes Target Specific Validation Rules 111 to validate the data structures in the application. For example, a Java target application may use Target Specific Validation Rules 111 to validate the Java objects contained in the application.

Other Target Platforms 120, such as the ASCII File Platform 121 and the XML Platform 130, pertain to data representation. In these cases, a target application may be any application that processes such data and utilizes Target Specific Validation Rules 111 to validate its contents. Note that in this context, target applications are not constrained to one particular programming language. For example, a Visual Basic application that processes an XML file and a Java application that processes an XML file may both be target applications within the XML Platform. The Javascript Platform 127 pertains to HTML pages rendered inside web browsers, in which case a target application may be any set of HTML pages that utilize Target Specific Validation Rules 111 in Javascript to validate the data entered in the HTML pages.

The Relational Database Platform 128 refers to relational database applications, in which case a target application may be an instance of a relational database that uses Target Specific Validation Rules 111 in SQL to validate the data contained in the database.

The Target Platforms 120 depicted are meant to be illustrative rather than exhaustive. Target Platforms for other existing programming languages and technologies, as well as for future programming languages and technologies, may be supported by the architecture.

Data in the system may be organized by attribute, entity, and domain. An attribute may be an individual data element, and an entity may be a group of one or more related attributes. For example, a Customer entity might contain attributes for Street Address, City, State, and Zip Code. A domain may be a group of entities and their associated attributes. A meta domain is an abstract group of entities and attributes that are not defined in any particular target application. Alternatively, a target domain is a group of entities and attributes that have concrete representations in a particular target application, for example, they may represent classes and instance variables in a Java program, tables and columns in a relational database, etc..

Validation logic may be embodied in metarules that are typically defined on the attributes and entities of a meta domain. Metarules may be, for example, Boolean logic constructs designed to determine the validity of an attribute or attributes. In other words, a validation metarule may be any logic that evaluates the value of one or more attributes and returns true or false. Aspects of the system of the present disclosure may utilize GUI tools, proprietary 4GLs, or a combination of both to define metarules.

There are many different types of validation metarules. Atomic metarules may analyze the value of a specific attribute in isolation from the other attributes of the meta domain. For example, atomic metarules may be capable of performing the following types of validations:

Required Validation – verifying that the attribute has a value.

Length Validation – verifying that the length of the attribute's value is within a defined minimum and maximum length.

Data Type Validation – verifying that the characters included in the value of the attribute belong to a defined set (e.g. the data type is numeric, the data type is alphanumeric, etc.).

Range Validation – verifying that the value of the attribute is within a defined range of values (e.g. a numeric value may be required to be between 0 and 100).

Enumeration Validation – verifying that the value of the attribute is a member of a defined set of values (e.g. a set of two character state codes).

Regular Expression Validation – verifying that the value of the attribute conforms to a specified regular expression.

Cross-attribute metarules may validate the relationship between the values of two or more attributes (e.g. is the value of "Attribute A" less than the value of "Attribute B"). Aggregation metarules may validate the aggregate value of an attribute for a collection of entities (e.g. the total of the values for an attribute cannot exceed a specified amount, or the average value of an attribute must exceed some specified amount). Conditional metarules may perform a validation when a defined set of preconditions are met (e.g. if "Attribute A" has a particular value and "Attribute B" has a particular value then "Attribute C" is required). Many metarules may fall into one or more of the above categories. However, since validation metarules can be arbitrarily complex, some metarules may defy any classification scheme. FIG 2 illustrates the basic architecture of the Validator System 110 according to one aspect of the system of the present disclosure. The Meta Domain Management System 210 may be responsible for the creation and management of meta domains and their associated metarules. The Meta Domain Management System 210 includes the following functional components: Meta Domain Administrator 211, Meta Domain Editor 212, Meta Domain Tester 213, Meta Domain Publisher 214, Meta Domain Exchange 215, and Domain Repository 216.

The Meta Domain Administrator 211 is responsible for authentication, authorization, and multi-user access. Authentication identifies users and verifies that they are permitted to use the Meta Domain Management System 210. Authorization verifies that users have the necessary privileges to perform the various functions of the Meta Domain Management System 210. Multi-user access manages the potential resource conflicts arising from multiple users accessing the system at the same time.

The Meta Domain Editor 212 is responsible for the creation, modification, and deletion of meta domains and their associated entities, attributes, and entity relationships.

The Meta Domain Editor 212 is also responsible for the creation, modification, and deletion of metarules using proprietary GUI tools and/or a 4GL. The Meta Domain Editor 212 is responsible for saving and retrieving meta domains and their associated metarules in cooperation with the Domain Repository 216. The Meta Domain Editor 212 also allows for the ad hoc testing of individual metarules and the immediate reporting of the results. These tests serve as an aid in the creation and modification of metarules.

FIG 3 shows the information contained in a meta domain according to one aspect of the system of the present disclosure. For each meta domain, the Meta Domains table 310 defines the name of the meta domain and the description of the meta domain. For each entity, the Meta Domain Entities table 320 defines the name of the domain containing the entity, the name of the entity, and the description of the entity. For each attribute, the Meta Domain Attributes table 330 defines the name of the domain and entity containing the attribute, the name of the attribute, and the description of the attribute. The Meta Domain Relationships table 340 describes the relationships between entities in a meta domain. In one aspect of the system of the present disclosure, for each relationship, the table defines the two entities involved, a description of the relationship, and the type of relationship (e.g. one-to-one, one-to-many, many-to-many).

FIG 4 shows the information contained in metarules according to one aspect of the system of the present disclosure. For each metarule, the Metarules table 410 defines the name of the metarule, various validation parameters used to validate the value of attribute(s) associated with the metarule, and an error message to display when the attribute fails validation. The validation parameters depicted in the Metarules table 410 are: whether the attribute is required ('Required'), the data type of the attribute ('Data Type'), the minimum length of the attribute ('Min Length'), the maximum length of the attribute ('Max Length'), and the valid values of the attribute ('Valid Values'). The validation parameters in the Metarules table 410 represent one aspect of the system of the present disclosure; other aspects of the system of the present disclosure may contain different validation parameters. The Association of Meta Domain Attributes to Metarules table 420 associates attributes to metarules. Each row associates the attribute that is uniquely identified by the names in the domain, entity, and attribute columns with the

metarule named in the metarule column. Metarules are reusable, and the same metarule can be associated with more than one attribute.

The Meta Domain Tester 213 is responsible for the creation, modification, deletion, and execution of tests for a particular meta domain. Tests are grouped together into test suites. Each test includes data values for the attributes of an entity or entities, and a specification of the expected results. The Meta Domain Tester 213 is capable of executing test suites and reporting the results of the test suite execution. The Meta Domain Tester 213 is also responsible for saving and retrieving test suites and their results in cooperation with the Domain Repository 216.

FIG 5 shows the information contained in a meta domain test suite according to one aspect of the system of the present disclosure. The Meta Domain Test Suites table 510 contains the name of the test suite and the meta domain on which the test suite is defined. The Meta Domain Tests table 520 enumerates the tests contained in the test suite. A test may be associated with more than one test suite. The Meta Domain Test Values table 530 defines the values for the attributes involved in the tests. A test involves one or more related entities. Note, however, that a particular entity may have multiple instances involved in the test. For example, a Customer entity may have multiple addresses based on an Address entity. One instance of the Address entity might be used for a home address while another instance of the Address entity might be used for a shipping address. Thus each row in the Meta Domain Test Values table 530 defines the value for an attribute of a particular instance of an entity participating in a specified test. The Meta Domain Expected Violations table 540 defines the expected results for a test. A violation is a metarule that failed (i.e. returned false) due to an invalid test value. Each test may produce zero, one, or many violations depending upon the validity of the test values. Each row in the Meta Domain Expected Violations table 540 identifies an attribute involved in a test and an associated metarule that is expected to fail based upon the test value defined for the attribute.

The Meta Domain Publisher 214 is responsible for the creation of documentation about a meta domain and its associated metarules. It is also responsible for saving and retrieving meta domain documentation in cooperation with the Domain Repository 216.

The meta domain documentation may be made available in a number of formats, for example, Microsoft Word documents, PDF documents, and HTML pages.

FIG 6 illustrates a possible structure of documentation for the meta domain according to one aspect of the system of the present disclosure. For each entity in the meta domain, the description of the entity is presented followed by the documentation for the attributes contained in the entity. For each attribute, the description of the attribute may be presented followed by a description of the validations associated with the attribute. The descriptions of the validations can be constructed using the error messages of the metarules associated with the attribute.

The Meta Domain Exchange 215 is responsible for exporting meta domains and their associated metarules to external systems. It is also responsible for importing meta domains and their associated metarules from external systems. In one aspect of the system of the present disclosure, the import/export of metarules may be accomplished, for example, using an XML representation of the metarules. Other aspects of the system of the present disclosure may use other representations.

The Domain Repository 216 is responsible for the persistent storage of meta domains, metarules, target domains, and the mappings between meta domains and target domains. It is also responsible for the persistent storage of the entities, attributes, tests, test suites, and documentation associated with domains. Different aspects of the system of the present disclosure may implement the Domain Repository 216 using different persistent storage mechanisms (e.g. XML file, relational database, etc.). The Domain Repository 216 provides an interface to the other components of the system that insulates them from the specific implementation of the Domain Repository 216.

The Validator System 110 is capable of supporting a broad range of Target Platforms 120, for example, ASCII File Platform 121, Cobol Platform 122, C Platform 123, C# Platform 124, C++ Platform 125, Java Platform 126, Javascript Platform 127, Relational Database Platform 128, Visual Basic Platform 129, and XML Platform 130. A Target Platform Adaptor 220 allows the system to generate Target Specific Validation Rules 111 from the metarules. Target Platform Adaptors 220 may include the following functional components: Target Domain Editor 221, Target Domain Rule Generator 222, Target Domain Tester 223, and Target Domain Publisher 224.

The Target Domain Editor 221 is responsible for the creation, modification, and deletion of the entities and attributes in the target domain. The entities and attributes may be created manually using the Target Domain Editor 221. Alternatively, depending upon the platform, the Target Domain Editor 221 may be able to import the entity and attribute definitions directly from the target application. For example, a Target Domain Editor 221 for a relational database may read the data dictionary of a database and utilize information about the database tables and columns to create target domain entities and attributes respectively. Other Target Domain Editors 221 may parse source code to gather information that may be used to create target domain entities and attributes. For example, the classes and instance variables of a Java application may be used to create target domain entities and attributes. Multiple target applications may reside on the same computer system, each having its own target domain.

The Target Domain Editor 221 is also responsible for mapping the entities and attributes in the meta domain to the entities and attributes in the target domain. The entities and attributes contained in the target application may differ from the definition of entities and attributes in the meta domain. For example, in the meta domain address attributes such as Street Address, City, State, and Zip Code may be defined on a Customer entity. In the target application, however, the address attributes may be defined on an Address entity and the Customer entity may reference the Address entity rather than defining the address attributes directly. Consequently, it may be necessary to map the entities and attributes defined in the meta domain to the entities and attributes in the target domain.

The Target Domain Editor 221 allows for the creation, modification, and deletion of target domain metarules. All of the validation rules used in a target application may not be available as metarules in the meta domain. The target application may even have entities and attributes for which there are no corresponding entities and attributes in the meta domain. In order to compensate for what might be lacking in the meta domain and its associated metarules, the Target Domain Editor 221 allows metarules to be associated directly with the attributes of the target domain. These metarules which are associated directly with the target domain are knows as target domain metarules. These target domain metarules may not be defined in the programming language of the target

application, but are defined using the same GUI tools and/or 4GL as the meta domain metarules. One distinction between target domain metarules and meta domain metarules may be that target domain metarules are associated with the entities and attributes of the target domain rather than being associated with the entities and attributes of the meta domain.

The Target Domain Editor 221 also allows for the creation, modification, and deletion of rulesets. A ruleset is a subset of the metarules available in the target domain. Different functionalities within an application may require different validation logic. Since all validation rules may not be suitable for all purposes, it is necessary to have the ability to group metarules into sets that can be managed, generated, and executed together.

Finally, the Target Domain Editor 221 is responsible for saving and retrieving target domains, target rules, and rulesets in cooperation with the Domain Repository 216.

FIG 7 shows the information contained in a target Java domain according to one aspect of the system of the present disclosure. For each Java target domain, the Java Domains table 710 defines the name of the target domain and the description of the domain. For each class, the Java Domain Classes table 720 defines the name of the domain containing the class, the name of the class, and the description of the class. For each instance variable, the Java Domain Instance Variables table 730 defines the name of the domain and class containing the instance variable, the name of the instance variable, the data type of the instance variable, the accessor method for obtaining the value of the instance variable, and the description of the instance variable. For instance variables that reference other object(s), the data type field defines the class of the referenced object(s). The Java Domain Object References table 740 extends the definition of those instance variables that reference other objects. The table defines the cardinality of the reference (i.e. whether the instance variable references a single object or multiple objects). It also defines whether or not the referenced object should be validated when the object containing the reference is validated.

FIG 8 shows the mapping of a meta domain to a Java target domain according to one aspect of the system of the present disclosure. Each row of the Mapping of Meta

Domain to Java Domain table 810 maps a meta domain attribute to a Java target domain instance variable.

The Target Domain Rule Generator 222 uses information contained in the meta domain, metarules, target domain, and target mapping to generate Target Specific Validation Rules 111 that are capable of executing within the environment of the target application. The Target Specific Validation Rules 111 are functionally equivalent to the metarules. The Target Domain Rule Generator 222 may generate validation rules for all the metarules or for a defined ruleset. The Target Specific Validation Rules 111 provide a developer working on the target application the ability to validate a target entity or entities and receive a list of validation violations. The Target Specific Validation Rules 111 generated by the Validator System 110 also allow for augmentation with validation rules written by a developer in the programming language of the target application.

The Target Domain Tester 223 is responsible for the creation, modification, deletion, and execution of tests for a particular target domain. Tests are grouped together into test suites. Each test contains data values for the attributes of an entity or entities, and a specification of the expected results. The Target Domain Tester 223 is capable of executing test suites and reporting the results of the test suite execution. It is also capable of using the Target Mapping to convert meta domain tests and test suites into target domain tests and test suites. The Target Domain Tester 223 is also responsible for saving and retrieving test suites and their results in cooperation with the Domain Repository 216.

FIG 9 shows the information contained in a Java target domain test suite according to one aspect of the system of the present disclosure. The Java Domain Test Suites table 910 contains the name of the test suite and the Java domain on which the test suite is defined. The Java Domain Tests table 920 enumerates the tests contained in the test suite. Note that a test may be associated with more than one test suite. The Java Domain Test Values table 930 defines the values for the instance variables involved in the tests. A test may involve one or more related classes. Note, however, that a particular class may have multiple instances involved in the test. For example, a Customer instance may reference multiple address instances based on an Address class. One instance of the Address class might be used for a home address while another instance of the Address class might be used for a shipping address. Thus each row in the

Java Domain Test Values table 930 defines the value for an instance variable of a particular instance of a class participating in a specified test. The Java Domain Expected Violations table 940 defines the expected results for a test. A violation is a metarule that failed (i.e. returned false) due to an invalid test value. Each test may produce zero, one, or many violations depending upon the validity of the test values. Each row in the Java Domain Expected Violations table 940 identifies an instance variable involved in a test and an associated metarule that is expected to fail based upon the test value defined for the instance variable.

The Target Domain Publisher 224 is responsible for the creation of documentation about a target domain and its associated validation rules. It is also responsible for saving and retrieving target domain documentation in cooperation with the Domain Repository 216, The target domain documentation may be made available in several formats, including but not limited to Microsoft Word documents, PDF documents, and HTML pages.

FIG 10 illustrates a possible structure of documentation for a Java target domain according to one aspect of the system of the present disclosure. For each class in the target domain, the description of the class may be presented followed by the documentation for the instance variables contained in the class. For each instance variable, the description of the instance variable may be presented followed by descriptions of the validations associated with the instance variable. The descriptions of the validations may be constructed using the error messages of the metarules associated with the attribute.

CREATION OF VALIDATION RULES

A process for creating validation rules according to one aspect of the system of the present disclosure is illustrated in FIG 11. The artifacts which may be created are shown to the right of the step that creates them. This process will be explored below. This example is for the purpose of illustrating the concepts embodied in the system of the present disclosure. It represents one aspect of the system of the present disclosure. Many other more complex aspects of the system of the present disclosure are possible.

The ordering of the steps in the process flow represents one aspect of the system of the present disclosure as there is flexibility in the order in which steps may be executed. In one example, a user may create a target domain before creating the metarules. In another example, a user may complete the steps shown and then repeatedly return to add more metarules and regenerate the target validation rules.

In the example shown in FIG 11, the first step in creating validation rules is for a user to manually create a meta domain, which may include entities and their associated attributes, using the Meta Domain Editor 212 (Step S1102). The meta domain may not be created with the target domain(s) in mind, and it may be that one or more of the target domains may not even be envisioned at the time the meta domain is created.

FIG 12 displays an example meta domain defined in the Example 1 Meta Domain table 1210 according to one aspect of the system of the present disclosure. The Example 1 Meta Domain has a single entity, Customer, defined in the Example 1 Meta Domain Entities table 1220. The Customer entity has nine attributes defined in the Example 1 Meta Domain Attributes table 1230: Last Name, First Name, Middle Initial, Home Phone, Work Phone, Street Address, City, State, and Zip Code. The user defines a name and a description for each domain, entity, and attribute. Since, in this example, there is only one entity, the Example 1 Meta Domain Relationships table 1240 has no entries.

After the meta domain is created, the user manually creates metarules using the Meta Domain Editor 212 (Step S1103). FIG 13 displays a representation of metarules created using the Meta Domain Editor 212. The Example 1 Metarules table 1310 depicts eight metarules: Last Name Rule, First Name Rule, Phone Rule, Required Rule, Street Address Rule, City Rule, State Rule, and Zip Code Rule. Each metarule contains data or logic specific to the type of metarule. In addition, each metarule contains an error message to be displayed when the validation fails.

The metarules are defined on the attributes of the meta domain. The Association of Example 1 Meta Domain Attributes to Metarules table 1320 shows the association of the metarules to the attributes in the meta domain. The relationship between metarules and attributes is many-to-many. A metarule can be associated with more than one attribute (e.g. a metarule can be created for one attribute and reused for other attributes). For example, the Phone Rule is associated with both the Home Phone attribute and the

Work Phone attribute. Similarly, an attribute can have zero, one, or any number of metarules associated with it. In the example shown in FIG 13, the Middle Initial attribute has no metarules associated with it, while the Home Phone attribute is associated with both the Phone Rule and the Required Rule.

After the metarules are created, the user may identify a target application and create a target domain for the application using the Target Domain Editor 221 (Step S1104). The target domain may include concepts that correspond to the entity and attribute concepts of the meta domain. For example, if the target application is a relational database, then the tables and columns of the relational database may represent entities and attributes in the target domain respectively. If the target application is a Java program, then the classes and instance variables of the Java program may represent entities and attributes in the target domain. The entities and attributes in the target domain may have names and descriptions identical to the entities and attributes in the meta domain. In addition, the target domain entities and attributes may contain other types of information specific to a particular target domain.

The user may use the Target Domain Editor 221 to manually create a target domain by inputting all the target domain entities, attributes, and related information. In some cases, however, the Target Domain Editor 221 may be able to programmatically create the target domain using information obtained from the target application (e.g. reading the information contained in the data dictionary of a relational database, parsing the source code of a target application, etc.).

FIG 14 shows an example target domain containing the Person and Address classes of a target Java application according to one aspect of the system of the present disclosure. The Example 1 Java Domain is defined in the Example 1 Java Domain table 1410. The classes making up the target domain, Person and Address, are defined in the Example 1 Java Domain Entities table 1420. The instance variables contained in the classes are defined in the Example 1 Java Domain Instance Variables table 1430. The Address class contains the instance variables street, city, state, and zip. The Person class contains the instance variables lastName, firstName, middleInitial, homePhone, workPhone, and homeAddress (a reference to an object of the Address class). Each instance variable in the target domain has a name, data type, accessor method name, and

description. The data type for the homeAddress instance variable is Address since it is a reference to an object of the Address class. The Example 1 Java Domain Object References table 1440 has additional information about the homeAddress instance variable. The table defines the cardinality of the reference, one, and whether or not the referenced object should be validated along with the Person object, yes.

The entities and attributes in the target domain may not correspond one-to-one with the entities and attributes in the meta domain. After the target domain is created, the meta domain may be mapped to the target domain (i.e. the attributes in the meta domain may be associated with attributes in the target domain). This mapping may be performed manually by the user using the Target Domain Editor 221 (Step S1105). Note that not every attribute in the meta domain may be mapped to an attribute in the target domain. Similarly, not every attribute in the target domain may have a mapping from an attribute in the meta domain. In other words, the attributes in the meta domain and the attributes in the target domain may not have a one-to-one correspondence.

FIG 15 shows the mapping of the Example 1 meta domain to the Example 1 Java domain according to one aspect of the system of the present disclosure. Note that the homeAddress instance variable in the Person class does not have a mapping from the meta domain since no such relationship exists in the meta domain.

Once the target mapping is complete, the Target Domain Rule Generator 222 is used to generate the validation rules using the information contained in the meta domain, metarules, and target mapping (Step S1106). FIG 16 illustrates the basic process flow for rule generation according to one aspect of the system of the present disclosure 1600. First, each metarule is examined to determine if it is associated with a target domain attribute (Step S1610). A metarule is associated with a target domain attribute if it is associated with a meta domain attribute that is mapped to a target domain attribute or if it is associated directly with a target domain attribute. For each metarule associated with a target domain attribute, a target domain metarule validation function is generated that contains the logic for that metarule (Step S1620). The target domain metarule validation function takes the value of a target domain attribute as an argument and returns a validation violation if the value is not valid. Once logic is generated for each applicable metarule, each entity in the target domain is examined (Step 1640). For each attribute

contained in the entity (Step S1650), a target domain attribute validation function may be generated (Step S1660). A target domain attribute validation function may take an attribute value as an argument and may return a data structure containing the validation violations for the attribute if the value is not valid. Once the target domain attribute validation functions are generated, a target domain entity validation function may be generated for the entity (Step S1680). A target domain entity validation function may take an entity as an argument and may return a data structure containing the validation violations for the entity if the entity contains data that is not valid.

The process for generating a target domain attribute validation function 1660 may involve generating a function call to the corresponding metarule validation function for each metarule associated with the target domain attribute (Steps S1661 – S1663). These function calls invoke the metarule validation functions that were previously generated, passing the attribute's value as an argument and placing any returned violations into a data structure that is returned by the target domain attribute validation function. The process for generating a target domain entity validation function 1680 may involve generating a function call to the corresponding target domain attribute validation function for each attribute contained in the entity (Steps S1681 – S1683). These function calls invoke the target domain attribute validation functions that were previously generated, passing the attribute's value as an argument and placing any returned violations into a data structure that is returned by the target domain entity validation function.

FIG 17 shows the Example 1 Java Specific Validation Rules 1715 that were generated by the Target Domain Rule Generator 222. Each box represents a validation function. The metarule validation functions generated for the metarules are validateLastNameRule 1726, validateFirstNameRule 1727, validateRequiredRule 1728, validatePhoneRule 1729, validateStreetAddressRule 1735, validateCityRule 1736, validateStateRule 1737, and validateZipCodeRule 1738. The target domain attribute validation functions generated for the target domain attributes are validateLastNameAttribute 1721, validateFirstNameAttribute 1722, validateHomePhoneAttribute 1723, validateWorkPhoneAttribute 1724, validateHomeAddressAttribute 1725, validateStreetAttribute 1731, validateCityAttribute 1732, validateStateAttribute 1733, and validateZipAttribute 1734. The target domain

entity validation functions generated for the target domain entities are validatePersonObject 1720 and validateAddressObject 1730. The lines indicate the calling sequence or flow of execution of the validation. For example, the validatePersonObject function 1720 first calls the validateLastNameAttribute function 1721, passing the value of the lastName instance variable as an argument and receiving any violations as the return value. Note that since the Home Phone attribute in the meta domain has two metarules associated with it, the Phone Rule and the Required Rule, the validateHomePhoneAttribute function 1723 calls two different metarule validation functions for the homePhone instance variable, validateRequiredRule 1728 and validatePhoneRule 1729. Also note that since the homeAddress instance variable is an object reference to an Address object and the value of the validate column in the Example 1 Java Domain References table 1440 is yes, the Person object is responsible for validating the Address object referenced by its homeAddress instance variable. Therefore the validateHomeAddressAttribute function 1725 invokes the validateAddressObject function 1730 to validate its Address object. The validateAddressObject function 1730 is responsible for validating the instance variables contained in the Address object and returning a collection of violations to the validateHomeAddressAttribute function 1725. Finally, note that there is no target domain attribute validation function for the middleInitial instance variable since there are no metarules associated with this attribute.

In Step S1107 the system integrates the Java validations rules into the rest of the Java application . Given a customer object in need of validation, the Example 1 Java Application 1710 makes a function call to validatePerson 1720, passing the customer object as an argument and receiving a collection of violations as a return value.

CREATION OF TESTS

Test suites contain one or more tests. Each test defines values for the attributes of one or more related entities and the expected results of the test. Meta domain test suites and tests are created using the Meta Domain Tester 213. Target domain test suites and tests are created using the Target Domain Tester 223.

FIG 18 illustrates a test suite for the Example 1 meta domain. The Example 1 Test Suite is defined in the Example 1 Meta Domain Test Suites table 1810. This test suite contains two tests, Test 1 and Test 2, which are defined in the Example 1 Meta Domain Tests table 1820. The data for the two tests are defined in the Example 1 Meta Domain Test Values table 1830. Note that since all the data values defined for Test 1 are valid, the Example 1 Meta Domain Expected Violations table 1840 contains no violations for Test 1. Test 2, on the other hand, contains invalid data values for Work Phone, State, and Zip Code. Hence the Example 1 Meta Domain Expected Violations table 1840 contains violations for the Phone Rule, State Rule, and Zip Code Rule. A test is successful when the violations returned match the expected violations. A test that returns no errors is only successful if all of the test values in the test are valid.

FIG 19 illustrates a test suite for the Example 1 Java domain. The Example 1 Test Suite is defined in the Example 1 Java Domain Test Suites table 1910. This test suite contains two tests, Test 1 and Test 2, which are defined in the Example 1 Java Domain Tests table 1920. The data for the two tests are defined in the Example 1 Java Domain Test Values table 1930. Note that in both tests, the homeAddress instance variable refers to the Address 1 instance of Address. Furthermore, note that since all the data values defined for Test 1 are valid, the Example 1 Java Domain Expected Violations table 1940 contains no violations for Test 1. Test 2, on the other hand, contains invalid data values for workPhone, state, and zip. Hence the Example 1 Java Domain Expected Violations table 1940 contains violations for the Phone Rule, State Rule, and Zip Code Rule. A test is successful when the violations returned match the expected violations. A test that returns no violations is only successful if all of the test values in the test are valid.

FIG 20 is a diagram illustrating the process for executing test suites. Each test in the test suite is executed in succession (Steps S2002 - S2013). Each entity in the test is validated (Steps S2003 - S2009) by validating its attributes (Steps S2004 - S2008). Each attribute has its test value validated by the metarules associated with the attribute (Steps S2005 - S2007). All of the violations detected by Step S2006 (i.e. the metarules that fail) are ultimately returned to Step S2010 once the test is complete. The detected violations are compared to the expected violations (Steps S2010). If the detected violations match

the expected violations, then the success of the test is reported (Step S2011). Otherwise, test failure is reported along with the differences between the detected violations and the expected violations (Step S2012).

CREATION OF DOCUMENTATION

Documentation for a meta domain is created using the Meta Domain Publisher 214. Documentation for a target domain is generated using a Target Domain Publisher 224. The process flow for generating documentation is the same regardless of the domain (i.e. meta domain or target domain). FIG 21 illustrates the basic process for creating documentation for a domain according to one aspect of the system of the present disclosure. First, the description of the domain is documented (Step S2102). Next, documentation is generated for each entity in the domain (Steps S2103 - S2111). For a given entity, the description of the entity is documented (Step S2104). Then each attribute in the entity is documented (Steps S2105 - S2110). For a given attribute, the documentation begins with the description of the attribute (Step S2106) followed by documentation for each metarule associated with the attribute (Steps S2107 - S2109) according to one aspect of the system of the present disclosure. FIG 22 displays sample documentation generated for the Example 1 meta domain. FIG 23 displays sample documentation generated for the Example 1 Java domain according to one aspect of the system of the present disclosure.

EXAMPLE

FIG 24 depicts how the system of the present disclosure can be utilized within an enterprise according to one aspect of the system of the present disclosure. Each meta domain and its associated metarules can be used to generate validation rules for multiple target applications within the enterprise. First a meta domain is created (Step S2402). Then the metarules associated with the meta domain are created (Step S2403). For each target application (Step S2404), a target domain is created (Step S2405). Next, the meta domain is mapped to the target domain (Step S2406). Then the target validation rules are generated (Step S2407) and integrated with the target application (Step S2408). Multiple meta domains could be created for different application domains within the enterprise. In

that case, the above process would be repeated for each meta domain. It is even possible that a single target application could have multiple meta domains mapped to it, each generating validation rules for the target application.

The system of the present disclosure can also be used between enterprises. An enterprise could import validation requirements from an external enterprise such as a standards body. Alternatively, an enterprise could export validation requirements to its business partners to promote standardization of validation logic. FIG 25 illustrates a typical workflow for two organizations sharing validation logic according to one aspect of the system of the present disclosure. Enterprise A 2510 creates a meta domain (Step S2512) and metarules (Step S2513). Enterprise B 2520 imports the meta domain and metarules from Enterprise A 2510 (Step S2521). For each target application (Step S2522), Enterprise B 2520 creates a target domain (Step S2523) and maps the meta domain to the target domain (Step S2524). Finally, the target validation rules are generated (Step S2525) and integrated with the target application (Step S2526).